AFRL-RY-WP-TR-2020-0250

QUANTITATIVE METRIC AND AUTOMATED TOOLSET FOR OBFUSCATED LOGIC SECURITY EVALUATION

Yier Jin
University of Florida

David Pan
University of Texas at Austin

AUGUST 2020
Final Report

STINFO COPY

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
August 2020	Final	21 February 2018 – 30 December 2019

4. TITLE AND SUBTITLE
QUANTITATIVE METRIC AND AUTOMATED TOOLSET FOR OBFUSCATED LOGIC SECURITY EVALUATION

5a. CONTRACT NUMBER
FA8650-18-1-7822

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62716E

6. AUTHOR(S)
Yier Jin (University of Florida)
David Pan (University of Texas at Austin)

5d. PROJECT NUMBER
N/A

5e. TASK NUMBER
N/A

5f. WORK UNIT NUMBER
Y1R1

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Florida
Gainesville, FL 32611

University of Texas at Austin
Austin, TX 78712

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
Sensors Directorate
Wright-Patterson Air Force Base, OH
45433-7320
Air Force Materiel Command
United States Air Force

Defense Advanced Research Projects Agency
DARPA/MTO
675 North Randolph Street
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/RYDT

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-RY-WP-TR-2020-0250

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with The Under Secretary of Defense memorandum dated 24 May 2010 and AFRL/DSO policy clarification email dated 13 January 2020. This material is based on research sponsored by the Air Force Research Lab (AFRL) and the Defense Advanced Research Agency (DARPA) under agreement number FA8650-18-2-7833. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes not withstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air force Research Laboratory (AFRL) and Defense Advanced Research Agency (DARPA) or the U.S. Government. Report contains color.

14. ABSTRACT
In this final report, we describe the project outcome, NEOS, a C++ software framework which implements the majority of state-of-art netlist obfuscation/deobfuscation algorithms. The API of NEOS allows for easy extension and modification of (de)obfuscation algorithms.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Pompei Orlando
Unclassified	Unclassified	Unclassified	SAR	8	19b. TELEPHONE NUMBER *(Include Area Code)* N/A

DARPA OMG Final Report: Quantitative Metric and Automated Toolset for Obfuscated Logic Security Evaluation

Yier Jin
Electrical and Computer Engineering Department
University of Florida
Gainesville, Florida
Email: yier.jin@ece.ufl.edu

David Pan
Electrical and Computer Engineering Department
University of Texas at Austin
Austin, Texas
Email: dpan@ece.utexas.edu

Abstract

In this final report, we describe the project outcome, NEOS, a C++ software framework which implements the majority of state-of-art netlist obfuscation/deobfuscation algorithms. The API of NEOS allows for easy extension and modification of (de)obfuscation algorithms.

I. Introduction

Due to the high costs of maintaining facilities for deeply scaled integrated-circuit manufacturing, semiconductor companies outsource fabrication to consolidated fabs. This has raised serious concerns regarding malicious modification of designs and reverse-engineering leading to IP theft. Logic locking thwarts these threats by adding programmability to the design, so that the IC will not be operational unless configured correctly post-fabrication by a secret key. Split-manufacturing is based on simply fabricating upper layers with larger feature sizes in a trusted facility. IC camouflaging is based on inserting nano-device structures in the fabricated IC that are difficult to reverse engineer by end-users.

While implementing these schemes requires different technologies and design/manufacturing flows, from an attacker's perspective, all these schemes (more true for locking/camouflaging) can be modeled with somewhat similar mathematics: as transforming a circuit $c_o(x)$ to $c_e(x,k)$ by adding l additional inputs/variables k called key-inputs which model the netlist's ambiguity to the attacker. Given this, the goal of the attacker is to recover c_o given c_e. An oracle-guided attacker is one that in addition to c_e has black-box access to c_o. In practice this means that the attacker has an operational IC that he/she can use to test chosen input patterns and observe outputs. Security of an obfuscation scheme heavily depends on the observability that the attacker has on the circuit. For example, without full scan-chain access, an oracle-guided attacker will have to reason about sequential behavior which is a much more complex computational task.

The development of better and better obfuscation schemes has been followed by the invention of more and more powerful attacks. Access to high-performance obfuscation and deobfuscation software is critical to research in these areas as it removes the need to build existing algorithms from scratch. With this in mind, this report presents a 25K-lines-of-code C++ object-oriented framework that implements a myriad of existing (de)obfuscation algorithms, along with an easy-to-use and extendable API that is discussed herein.

II. Netlist Representation

The first step of developing any netlist-based software tool is representing the netlist itself. NEOS supports two main objects/classes `circuit` and `aig` for storing netlists. Both use a custom C++ container to keep nodes that are accessed by unique IDs. The custom data-structure allows for dynamic removal of nodes while supporting array-like fast indexing. `circuit` supports hierarchical designs with primitive gates (and/nand/xor/xnor/...) and instances from a standard-cell library. Netlists in `circuit` are represented by an array of `gate` and `wire` objects that link to each other with fanout and fanin sets. `circuit` supports reading-in designs from a gate-level verilog (supporting ranged/indexed net definitions) and bench files. The read-in is based on parsers written in Flex/Bison. Key inputs are specified by starting the wire-name with `keyinput`.

`circuit` objects can be converted to `aig` objects which store structurally-hashed [1] And-Inverter-Graphs (AIG). AIGs, which are used in almost all modern verification/circuit-reasoning engines, are a great way to represent the circuit in a library-free manner with uniform nodes. AIGs allow for various simplification algorithms due to their semi-canonical nature. NEOS implements several of these including one-level hashing, k-cut-sweeping and DAG-aware rewriting [1].

NEOS supports various graph algorithms for AIGs and circuits. DAG algorithms including local/global-topological-sorting, feedback-arc-set, transitive-fanin/fanout are necessary to many (de)obfuscation algorithms which both objects support. These objects support editing operations such as removing and adding individual nodes as well as adding/inserting other objects of the same type (compounding). Another important category of DAG algorithms is partitioning. NEOS supports k-cut enumeration/detection which finds (fanout-free) logic cones in the circuit for both `circuit` and `aig` objects. This is crucial in an array of obfuscation schemes such as LUT-insertion since in LUT-insertion any k-cut can be replaced with a k-input LUT. In addition, NEOS supports equal-partitioning which will try to partition the circuit into similarly-sized sub-graphs. This is a great tool for performing density-minimizing obfuscation.

The netlist classes also include other important member functions such as combinational/sequential pattern simulation and error-checking.

III. Obfuscation Algorithms

Given the above netlist representation and graph algorithms, it is possible to build the majority of obfuscation schemes. Obfuscation algorithms are significantly less complicated than deobfuscation algorithms. It is possible to divide obfuscation algorithms to those studied before the invention of the oracle-guided SAT attack (traditional) and those that followed the SAT attack and try to specifically mitigate it (SAT-resilient).

The majority of traditional schemes are based on throwing various key-controlled/ambiguous elements into the circuit. Many various flavors of these schemes can be built using the netlist representation and the supported DAG algorithms, plus pattern simulation. NEOS implements an array of such traditional schemes including, random-X(N)OR/MUX/LUT insertion [2], insertion driven by error-rate cost-functions with hill-climbing, insertion in locations with maximum hamming-distance [3], and insertion of parity units (parity functions are hard to learn with shallow learners per computational learning theory). The hill-climbing obfuscation algorithm is a very important module as it can be used to optimize any given metric. This scheme is hence given its own object with a cost-function that can be replaced any C++ function which allows for easy extension.

A major part of the obfuscation-suite is dedicated to interconnect locking by inserting multiplexer (MUX) and tri-state-buffers. The interconnect locking suite supports cross-bar/MUX-network insertion [4] with different site-selection strategies (random, cost-function driven, k-cut, ...). The interconnect locking suite also includes a hill-climbing strategy which can be designed to maximize metrics such as feedback-arc-set size, graph density, minimum density for equal coverage of the netlist and so on [5].

Almost all SAT-resilient schemes rely on point-functions which are able to hide a small number of points in the circuit's truth-table from the attacker. These schemes try to achieve the Exact-Functional-Secrecy (EFS) notion of security per [6]. EFS security is defined loosely as: "the attacker should not be able to learn the precise functionality of the circuit in time t". This notion is stronger than key-recovery: "the attacker should not be able to recover the correct key in time less than t". It also allows for the attacker to approximate the circuit with exponentially good accuracy and is hence not suited for designs where approximation by attackers is a concern. As for EFS obfuscation, NEOS supports numerous schemes. Anti-SAT [7] with functional/structural obfuscation, DTL [8] with various insertion strategies, and corrupt-and-correct schemes [9].

While the above mentioned schemes are part of the body of research on obfuscation, we now know that the most secure form of EFS schemes is ones in which an existing (multi)-point-function in the circuit is detected and replaced with a look-up-table which NEOS supports [10]. These approaches are more secure since the reductive nature means that the attacker cannot simply remove added structures. One particular scheme that NEOS implements is based on using SAT-calls to identify all the input patterns that activate a particular net and if the number is smaller than a given size, then it is replaced with a look-up-table. In addition, NEOS supports the "larger-than-z" look-up-table from [6] which tricks the attacker into thinking that there are additional hidden patterns that need to be found by

querying the entire input space.

IV. Deobfuscation Algorithms

NEOS implements deobfuscation algorithms in an object-oriented and polymorphic way. The simplest (top in the class hierarchy) deobfuscation object is an oracle-less class which includes only the obfuscated circuit c_e. The object upon execution of the attack will return a correct key or a key hypothesis with probability values attached to the guessed key-bits. An oracle-guided attack object which derives from this topmost class has an additional oracle object c_o. This class will take care of linking the input-outputs of c_o and c_e and error-checking. c_e and c_o can be sequential for which a different class is used.

The interface of the attacks (obfuscated/oracle circuit as the input and and key-hypothesis as the output) allows for chaining together various attacks. This is particularly useful in attacks that can guarantee the correctness of some key bits which allow for simplifying the circuit and continuing with another or the same attack. The key-hypotheses can also be chained by weighted adding of probability values attached to keys.

A. Combinational Deobfuscation

In combinational deobfuscation the circuit c_o and c_e are both stateless. Oracle-guided attacks in literature have mostly focused on this case with the SAT-attack being the most prominent. The SAT attack has the unique ability to guarantee the recovery of a correct key if the original hypothesis model is expressive enough to include the original circuit (there exists k_* for which $c_e(x, k_*)$ agrees with $c_o(x)$ on all input patterns).

SAT attacks use SAT queries to guide the deobfuscation process. Similar to SAT-based verification or equivalence checking, in SAT attacks circuits are built to represent a specific condition and then converted to Conjunctive-Normal-Form (CNF) formulae. In the baseline SAT attack a mitter condition is formed by building the circuit $M \equiv (c_e(x, k_1) \neq c_e(x, k_2)$. If this circuit is converted to CNF and satisfied with a SAT solver, the $\hat{x}$ that is returned by the solver is called a discriminating input pattern (DIP). This input pattern if queried on the oracle, is guaranteed to trim the hypothesis key-space by at least one wrong key. Then the correct input-output observation condition is built as $(c_e(\hat{x}, k_1) = c_o(\hat{x}))$ which will be a new circuit and its output must be ANDed with M and the process is repeated.

NEOS implements various transforms for building and compounding such circuits/conditions. NEOS implements an API for converting circuits to SAT formulae. This conversion happens to have a relatively significant impact on the performance of SAT solving. Currently NEOS supports the baseline Tseitin transform in addition to the technology-mapping algorithm from [11]. Both techniques will add a set of clauses to a solver and producing a mapping between wires in the circuit and variables in the solver. An important feature which is very important for incrementally building and solving SAT problems is that the CNF conversion can take as input a set of pre-existing variables/wires, in which case the conversion will not create new variables for these nets. This for instance can be used to stitch a new copy of the circuit to an earlier CNF, connecting the input variables rather than duplicating them.

Currently NEOS interfaces to three SAT solvers Glucouse, MiniSAT, and CryptoMiniSAT with Glucouse being the default and preferred choice due to solver simplification and the support for copying solvers. Copying solvers is critical to many advanced deobfuscation routines which will fail until we add copy support to other solvers. The solvers are wrapped with a parent class which has a unified interface. This allows developers to add other solvers by simply replicating and implementing the solver API.

NEOS implements almost all existing SAT based deobfuscation attacks. In addition to the baseline SAT attack [12] NEOS implements AppSAT [8] which is a SAT based attack that is aware of the error rate of the hypothesis key and can hence be terminated earlier in cases where approximation is sufficient. It also implements (t)-DDIP and AppSAT termination conditions [8] which allow for early termination in certain cases. For instance, these termination conditions can signal termination when the error rate of an l-bit key is less than a small t.

The SAT based attacks are also implemented in an object-oriented manner with many optimization techniques that can be applied to any of the attacks. For instance various subkey extraction schemes are supported. First is the simple backbone-analysis [12] which given a condition on the key $\phi(k)$, will perform a SAT solver query on each key bit and determine if given $\phi(k)$ a particular key bit is resolved. A stronger subkey extraction routine is

the wire-disagreement analysis which can find multiple settled key-bits if they fall in the fanin cone of the same internal net in the circuit [8].

The synergy between circuits and Boolean conditions and CNF formulae means that simplifying circuits will result in simpler formulae and faster SAT solving. An `opt` module is included in NEOS which supports various simplification routines for both `circuit` and `aig` classes. `aig` simplification routines are faster due to the AIG-specific sweeping algorithms. NEOS currently supports SAT-sweeping which uses SAT solver calls to find equivalent nodes in the circuit and merges them. The `aig` class supports the fast Fraig SAT-sweeping algorithm [13] which is orders of magnitude faster than the `circuit` class's SAT-sweep which uses the baseline counter-example driven algorithm. Both netlist representations support BDD-sweeping which derives the BDD of internal nets up to a given BDD size bound and finds equivalent nodes. Note that only SAT-based simplification can use external conditions which are needed when simplifying circuits further given current input-output pairs observed in the attack. Simplifying key-conditions called key-condition-crunching (KC2) [14] can be applied to various attacks as well. Note that all sweeping techniques (equivalent node detection) are derived from the class `Sweep`. A separate analysis and commit stage allows for finding nodes but not merging them which is useful in many instances.

Cyclic SAT attacks are implemented in NEOS as well. These are attacks that can deobfuscate circuits that include intentional combinational cycles [15]. The IcySAT attacks [16] are the most recent and strongest algorithms for cyclic deobfuscation which are implemented in NEOS. Furthermore, the analysis step in the `Sweep` class can be used to find equivalent nodes and then merging them in reverse-topological order can create cyclic-yet-combinational circuits which are much harder to deobfuscate.

NEOS implements statistical attacks as well in the `stat` module. Most importantly the hill-climbing attack [17] which uses a simulated-annealing algorithm to optimize a key. The listing below shows a code snippet for reading in, obfuscating and deobfuscating the c432 benchmark.

```
circuit co; co.read_bench("./bench/c432.bench");
circuit ce = co;
enc::enc_xor_rand(ce, random_boolvec(32));
dec::sat_dec_exact(co, ce, iteration_limit);
```

B. Sequential Deobfuscation

In sequential deobfuscation c_e and c_o are sequential circuits. Sequential deobfuscation is a rather overlooked area with little tool support due to the increased complexity of the algorithms. In SAT-based sequential deobfuscation, the SAT query that searches for DIPs is replaced with a model-checking query that searches for a sequence of DIPs called DISs. NEOS supports sequential attacks using an external model-checker i.e. nuXmv. nuXmv implements state-of-the-art model-checking routines that can be tweaked to improve the attack.

Having an instance of a model-checker inside the framework so that it can be queried repeatedly without destroying the SAT-solvers can speed up sequential attacks by 100X [14]. NEOS hence implements several state-of-the-art model-checking routines through the polymorphic objects without the need for an external checker. The `Bmc` class implements bounded-model-checking (BMC) by unrolling and optional simplification at the circuit/AIG/SAT-solver level. Per Fig. 1, key-condition-crunching [14] techniques are integrated with sequential deobfsucation by keeping simplified versions of the unrolled circuit and further simplifying them as the condition on the key gets more constrained throughout the attack. Condition simplification is significantly more important in the problem of sequential deobfuscation, as sequential deobfuscation SAT formulae sizes grow at much higher rates. This is because as the discriminating input sequences get deeper, larger and larger circuit conditions need to be added to the solver. However, the added conditions have lots of redundancy due to the unrolling.

NEOS implements a sequential version of the hill-climbing attack which is surprisingly successful in approximating the functionality of large sequential circuits with user-provided parameters such as number of random patterns to query and maximum depth of exploration.

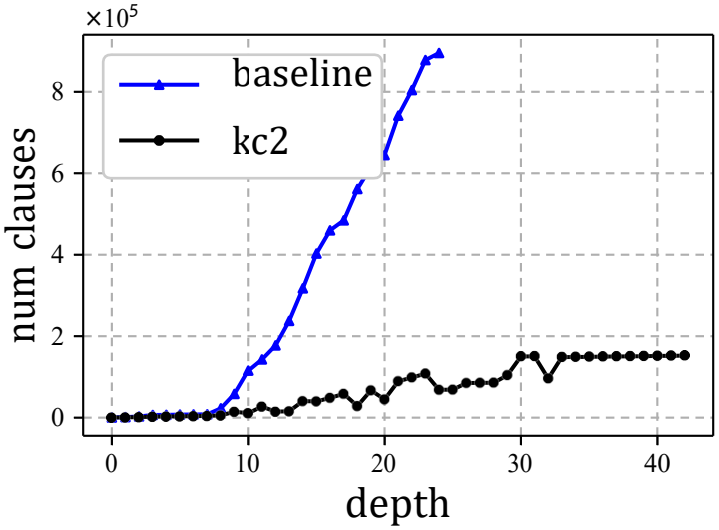


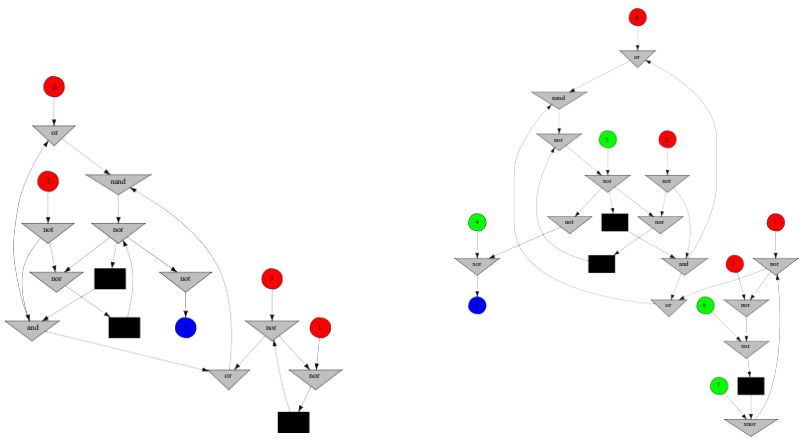
Fig. 1: Sequential SAT attack on the s400 benchmark with 36 key-bits with and without key-condition-crunching (KC2). KC2 enables reaching further into the state-space with low clause count.

V. Miscellaneous

NEOS implements a simple flow for oracle-less analysis. This involves a module for keeping track of how key-logic transforms with resynthesis. NEOS has an interface to ABC which is used for various tasks including model-checking, equivalence checking and simplification. ABC's routines for circuit simplification are a result of a decade of development at Berkeley and are the fastest when it comes to circuit compression.

NEOS interfaces to the `Cudd` BDD package which is used to translate circuits/AIGs into BDDs. This can be used to implement BDD based (de)obfuscation as well. The BDD package is also used to perform cube operations used in several aspects of obfuscation.

NEOS has a SAT-based implementation of REFSM [18] that uses an All-SAT routine to mine for states given a set of state-registers which can be used to deobfuscate FSM-based obfsucation. NEOS supports drawing circuits to a `.dot` graph format for visualization per Fig. 2.



(a) original circuit (b) obfuscated circuit

Fig. 2: NEOS obfuscating the s27 circuit. Green inputs are added keys and black-boxes are latches.

VI. Conclusion

In this report we described NEOS a C++ framework for netlist-level circuit obfuscation/deobfuscation. Deobfuscation attacks are complicated algorithms for which fast available implementations are rare. NEOS promises to fill this gap by providing an array of existing attacks and defenses for better understanding the problem of netlist locking/unlocking.

References

[1] Alan Mishchenko, Satrajit Chatterjee, and Robert Brayton, "Dag-aware aig rewriting a fresh look at combinational logic synthesis," in *Proceedings of the 43rd annual Design Automation Conference*. ACM, 2006, pp. 532–535.

[2] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending piracy of integrated circuits," in *Proc. Design, Automation and Test in Europe*, 2008, pp. 1069–1074.

[3] Jeyavijayan Rajendran, Huan Zhang, Chi Zhang, Garrett S Rose, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri, "Fault analysis-based logic encryption," vol. 64, no. 2, pp. 410–424, 2015.

[4] Kaveh Shamsi, Meng Li, David Pan, and Yier Jin, "Cross-lock: Dense layout-level interconnect locking using cross-bar architectures," in *GLSVLSI*, 2018.

[5] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z. Pan, and Yier Jin, "Cyclic obfuscation for creating sat-unresolvable circuits," in *GLSVLSI*, 2017, pp. 173–178.

[6] Kaveh Shamsi, David Z Pan, and Yier Jin, "On the impossibility of approximation-resilient circuit locking," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2019, pp. 161–170.

[7] Yang Xie and Ankur Srivastava, "Anti-sat: Mitigating sat attack on logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 2, pp. 199–207, 2018.

[8] Kaveh Shamsi, Travis Meade, Meng Li, David Pan, and Yier Jin, "On the approximation resiliency of logic locking and ic camouflaging schemes," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 14, no. 2, pp. 347–359, 2019.

[9] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proc. ACM Conf. on Computer & Communications Security*. ACM, 2017, pp. 1601–1618.

[10] Meng Li, Kaveh Shamsi, Travis Meade, Zheng Zhao, Bei Yu, Yier Jin, and David Z. Pan, "Provably secure camouflaging strategy for ic protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2018.

[11] Niklas Een, Alan Mishchenko, and Niklas Sörensson, "Applying logic synthesis for speeding up sat," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2007, pp. 272–286.

[12] Pramod Subramanyan, Sayak Ray, and Sharad Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, 2015, pp. 137–143.

[13] Alan Mishchenko, Satrajit Chatterjee, Roland Jiang, and Robert K Brayton, "Fraigs: A unifying representation for logic synthesis and verification," Tech. Rep., ERL Technical Report, 2005.

[14] Kaveh Shamsi, Meng Li, David Z Pan, and Yier Jin, "Kc2: Key-condition crunching for fast sequential circuit deobfuscation," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 534–539.

[15] Yuanqi Shen, You Li, Amin Rezaei, Shuyu Kong, David Dlott, and Hai Zhou, "Besat: behavioral sat-based attack on cyclic logic encryption," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. ACM, 2019, pp. 657–662.

[16] Kaveh Shamsi, David Z Pan, and Yier Jin, "Icysat: Improved sat-based attacks on cyclic locked circuits," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–7.

[17] Stephen M Plaza and Igor L Markov, "Solving the third-shift problem in ic piracy with test-aware logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 961–971, 2015.

[18] Travis Meade, Zheng Zhao, Shaojie Zhang, David Z. Pan, and Yier Jin, "Revisit sequential logic obfuscation: Attacks and defenses," in *The IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017.